AI + DS_Algo Assignment

Name: Aditya Arya

Roll no: 23CS3068

Problem 1:

Part A-

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<long long> arr(n + 1, 0), prefix(n + 1, 0);

    for (int i = 1; i <= n; i++) {
        cin >> arr[i];
        prefix[i] = prefix[i - 1] + arr[i];
    }

    int q;
    cin >> q;
    while (q--) {
        int l, r;
        cin >> l >> r;
        cout << prefix[r] - prefix[l - 1] << "\n";
    }

    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;
```

```
struct Point {
    double x;
    int y;
};

double gini(int c0, int c1) {
    int total = c0 + c1;
    if (total == 0) return 0.0;
    double p0 = (double)c0 / total;
    double p1 = (double)c1 / total;
    return 1.0 - (p0 * p0 + p1 * p1);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<Point> data(n);
    for (int i = 0; i < n; i++) cin >> data[i].x >> data[i].y;

    sort(data.begin(), data.end(), [](const Point &a, const Point &b) {
        return a.x < b.x;
    });

    int total0 = 0, total1 = 0;
    for (auto &p : data) {
        if (p.y == 0) total0++;
        else total1++;
    }

    int left0 = 0, left1 = 0;
    double bestThreshold = 0.0, bestGini = 1e9;

    for (int i = 0; i < n - 1; i++) {
        if (data[i].y == 0) left0++;
        else left1++;

        double threshold = (data[i].x + data[i + 1].x) / 2.0;
```

```
        int right0 = total0 - left0;
        int right1 = total1 - left1;

        int leftSize = left0 + left1;
        int rightSize = right0 + right1;

        double giniLeft = gini(left0, left1);
        double giniRight = gini(right0, right1);

        double weighted = (double)leftSize / n * giniLeft +
                          (double)rightSize / n * giniRight;

        if (weighted < bestGini) {
            bestGini = weighted;
            bestThreshold = threshold;
        }
    }

    cout << fixed << setprecision(6) << bestThreshold << " " << bestGini << "\n";
    return 0;
}
```

Problem 2:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    long long target;
    cin >> n >> target;

    vector<long long> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];

    int l = 0, r = n - 1;
```

```
    while (l < r) {
        long long sum = arr[l] + arr[r];
        if (sum == target) {
            cout << l + 1 << " " << r + 1 << "\n";
            return 0;
        }
        if (sum < target) l++;
        else r--;
    }

    cout << "-1 -1\n";
    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    int count0 = 0, count1 = 0;

    for (int i = 0; i < n; i++) {
        int y;
        cin >> y;
        if (y == 0) count0++;
        else count1++;
    }

    double entropy = 0.0;
    if (count0 > 0) {
        double p0 = (double)count0 / n;
        entropy -= p0 * log2(p0);
    }
    if (count1 > 0) {
        double p1 = (double)count1 / n;
```

```
        entropy -= p1 * log2(p1);
    }

    cout << fixed << setprecision(6) << entropy << "\n";
    return 0;
}
```

Problem 3:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];

    if (n == 0 || arr[0] == -1) return 0;

    TreeNode* root = new TreeNode(arr[0]);
    queue<TreeNode*> q;
    q.push(root);

    int i = 1;
    while (!q.empty() && i < n) {
        TreeNode* node = q.front();
        q.pop();

        if (i < n && arr[i] != -1) {
```

```
            node->left = new TreeNode(arr[i]);
            q.push(node->left);
        }
        i++;

        if (i < n && arr[i] != -1) {
            node->right = new TreeNode(arr[i]);
            q.push(node->right);
        }
        i++;
    }

    queue<TreeNode*> level;
    level.push(root);
    while (!level.empty()) {
        int sz = level.size();
        for (int j = 0; j < sz; j++) {
            TreeNode* node = level.front();
            level.pop();
            cout << node->val << " ";
            if (node->left) level.push(node->left);
            if (node->right) level.push(node->right);
        }
        cout << "\n";
    }

    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

double gini(int c0, int c1) {
    int total = c0 + c1;
    if (total == 0) return 0.0;
    double p0 = (double)c0 / total;
    double p1 = (double)c1 / total;
    return 1.0 - (p0 * p0 + p1 * p1);
}
```

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<pair<int,int>> data(n);
    for (int i = 0; i < n; i++) cin >> data[i].first >> data[i].second;

    int t;
    cin >> t;

    int left0 = 0, left1 = 0, right0 = 0, right1 = 0;
    for (auto &p : data) {
        if (p.first <= t) {
            if (p.second == 0) left0++;
            else left1++;
        } else {
            if (p.second == 0) right0++;
            else right1++;
        }
    }

    int leftSize = left0 + left1;
    int rightSize = right0 + right1;
    int total = leftSize + rightSize;

    double giniLeft = gini(left0, left1);
    double giniRight = gini(right0, right1);

    double weighted = (double)leftSize / total * giniLeft +
                      (double)rightSize / total * giniRight;

    cout << fixed << setprecision(6) << weighted << "\n";
    return 0;
}
```

Problem 4:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

struct TreeNode {
    long long val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(long long x) : val(x), left(NULL), right(NULL) {}
};

bool isBST(TreeNode* root, long long low, long long high) {
    if (!root) return true;
    if (root->val <= low || root->val >= high) return false;
    return isBST(root->left, low, root->val) && isBST(root->right, root->val, high);
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<long long> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];

    if (n == 0 || arr[0] == -1) {
        cout << "YES\n";
        return 0;
    }

    TreeNode* root = new TreeNode(arr[0]);
    queue<TreeNode*> q;
    q.push(root);

    int i = 1;
    while (!q.empty() && i < n) {
        TreeNode* node = q.front();
        q.pop();
```

```
        if (i < n && arr[i] != -1) {
            node->left = new TreeNode(arr[i]);
            q.push(node->left);
        }
        i++;

        if (i < n && arr[i] != -1) {
            node->right = new TreeNode(arr[i]);
            q.push(node->right);
        }
        i++;
    }

    if (isBST(root, LLONG_MIN, LLONG_MAX)) cout << "YES\n";
    else cout << "NO\n";

    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

struct Point {
    double x, y;
};

double variance(int count, double sumY, double sumY2) {
    if (count == 0) return 0.0;
    double mean = sumY / count;
    return sumY2 / count - mean * mean;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
```

```
vector<Point> data(n);
for (int i = 0; i < n; i++) cin >> data[i].x >> data[i].y;

sort(data.begin(), data.end(), [](auto &a, auto &b) {
    return a.x < b.x;
});

double totalSum = 0, totalSumSq = 0;
for (auto &p : data) {
    totalSum += p.y;
    totalSumSq += p.y * p.y;
}

double totalVar = variance(n, totalSum, totalSumSq);

double leftSum = 0, leftSumSq = 0;
int leftCount = 0;

double bestThreshold = 0.0, bestReduction = -1e18;

for (int i = 0; i < n - 1; i++) {
    leftSum += data[i].y;
    leftSumSq += data[i].y * data[i].y;
    leftCount++;

    double rightSum = totalSum - leftSum;
    double rightSumSq = totalSumSq - leftSumSq;
    int rightCount = n - leftCount;

    double varLeft = variance(leftCount, leftSum, leftSumSq);
    double varRight = variance(rightCount, rightSum, rightSumSq);

    double weighted = (double)leftCount / n * varLeft +
                      (double)rightCount / n * varRight;

    double reduction = totalVar - weighted;

    double threshold = (data[i].x + data[i + 1].x) / 2.0;

    if (reduction > bestReduction) {
```

```
            bestReduction = reduction;
            bestThreshold = threshold;
        }
    }

    cout << fixed << setprecision(6) << bestThreshold << " " << bestReduction << "\n";
    return 0;
}
```

Problem 5:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

TreeNode* buildTree(vector<int>& preorder, int preStart, int preEnd,
                    vector<int>& inorder, int inStart, int inEnd,
                    unordered_map<int,int>& inIndex) {
    if (preStart > preEnd || inStart > inEnd) return NULL;

    int rootVal = preorder[preStart];
    TreeNode* root = new TreeNode(rootVal);

    int idx = inIndex[rootVal];
    int leftSize = idx - inStart;

    root->left = buildTree(preorder, preStart + 1, preStart + leftSize,
                           inorder, inStart, idx - 1, inIndex);
    root->right = buildTree(preorder, preStart + leftSize + 1, preEnd,
                            inorder, idx + 1, inEnd, inIndex);

    return root;
}
```

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;

    vector<int> preorder(n), inorder(n);
    for (int i = 0; i < n; i++) cin >> preorder[i];
    for (int i = 0; i < n; i++) cin >> inorder[i];

    unordered_map<int,int> inIndex;
    for (int i = 0; i < n; i++) inIndex[inorder[i]] = i;

    TreeNode* root = buildTree(preorder, 0, n - 1, inorder, 0, n - 1, inIndex);

    queue<TreeNode*> q;
    q.push(root);
    while (!q.empty()) {
        int sz = q.size();
        for (int i = 0; i < sz; i++) {
            TreeNode* node = q.front(); q.pop();
            cout << node->val << " ";
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
    }
    cout << "\n";

    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

double entropy(const vector<long long>& counts) {
    long long total = 0;
    for (auto c : counts) total += c;
    if (total == 0) return 0.0;
```

```
    double H = 0.0;
    for (auto c : counts) {
        if (c == 0) continue;
        double p = (double)c / total;
        H -= p * (log(p) / log(2.0));
    }
    return H;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int C, K;
    cin >> C >> K;

    vector<vector<long long>> cat(K, vector<long long>(C));
    vector<long long> parent(C, 0);
    vector<long long> catTotals(K, 0);
    long long N = 0;

    for (int k = 0; k < K; ++k) {
        for (int c = 0; c < C; ++c) {
            cin >> cat[k][c];
            parent[c] += cat[k][c];
            catTotals[k] += cat[k][c];
        }
        N += catTotals[k];
    }

    double H_parent = entropy(parent);
    double weighted_child_entropy = 0.0;
    for (int k = 0; k < K; ++k) {
        if (catTotals[k] == 0) continue;
        double weight = (double)catTotals[k] / N;
        weighted_child_entropy += weight * entropy(cat[k]);
    }

    double info_gain = H_parent - weighted_child_entropy;
    cout << fixed << setprecision(6) << info_gain << "\n";
```

```
    return 0;
}
```

Problem 6:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

struct SparseTable {
    int n, K;
    vector<vector<int>> st;
    vector<int> log;

    SparseTable(const vector<int>& arr) {
        n = arr.size();
        K = log2(n) + 1;
        st.assign(n, vector<int>(K));
        log.assign(n + 1, 0);
        for (int i = 2; i <= n; i++) log[i] = log[i/2] + 1;
        for (int i = 0; i < n; i++) st[i][0] = arr[i];
        for (int j = 1; j < K; j++) {
            for (int i = 0; i + (1 << j) <= n; i++) {
                st[i][j] = min(st[i][j-1], st[i + (1 << (j-1))][j-1]);
            }
        }
    }

    int query(int L, int R) {
        int j = log[R - L + 1];
        return min(st[L][j], st[R - (1 << j) + 1][j]);
    }
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> arr(n);
```

```
    for (int i = 0; i < n; i++) cin >> arr[i];
    SparseTable st(arr);
    int q;
    cin >> q;
    while (q--) {
        int l, r;
        cin >> l >> r;
        l--, r--;
        cout << st.query(l, r) << "\n";
    }
    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    int cnt0 = 0, cnt1 = 0;
    for (int i = 0; i < n; i++) {
        int x; cin >> x;
        if (x == 0) cnt0++;
        else cnt1++;
    }
    if (cnt1 > cnt0) cout << 1;
    else cout << 0;
    return 0;
}
```

Problem 7:

Part A-

```
#include <bits/stdc++.h>
using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int v) : val(v), left(nullptr), right(nullptr) {}
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];
    if (n == 0 || arr[0] == -1) return 0;

    TreeNode* root = new TreeNode(arr[0]);
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;
    while (i < n) {
        TreeNode* node = q.front(); q.pop();
        if (i < n && arr[i] != -1) {
            node->left = new TreeNode(arr[i]);
            q.push(node->left);
        }
        i++;
        if (i < n && arr[i] != -1) {
            node->right = new TreeNode(arr[i]);
            q.push(node->right);
        }
        i++;
    }

    vector<int> inorder;
    stack<TreeNode*> st;
    TreeNode* curr = root;
    while (curr || !st.empty()) {
        while (curr) {
            st.push(curr);
```

```
            curr = curr->left;
        }
        curr = st.top(); st.pop();
        inorder.push_back(curr->val);
        curr = curr->right;
    }

    for (int j = 0; j < inorder.size(); j++) {
        if (j) cout << " ";
        cout << inorder[j];
    }
    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

double gini(const vector<int>& labels) {
    if (labels.empty()) return 0.0;
    int cnt0 = 0, cnt1 = 0;
    for (int y : labels) {
        if (y == 0) cnt0++;
        else cnt1++;
    }
    double p0 = (double)cnt0 / labels.size();
    double p1 = (double)cnt1 / labels.size();
    return 1.0 - p0 * p0 - p1 * p1;
}

double weightedGini(const vector<pair<int,int>>& data, int t) {
    vector<int> left, right;
    for (auto &p : data) {
        if (p.first <= t) left.push_back(p.second);
        else right.push_back(p.second);
    }
    double n = data.size();
    return (left.size() / n) * gini(left) + (right.size() / n) * gini(right);
}
```

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<pair<int,int>> data(n);
    for (int i = 0; i < n; i++) cin >> data[i].first >> data[i].second;

    int t1, t2;
    cin >> t1 >> t2;

    double g1 = weightedGini(data, t1);
    double g2 = weightedGini(data, t2);

    if (g1 < g2 || fabs(g1 - g2) < 1e-9)
        cout << t1 << " " << fixed << setprecision(6) << g1;
    else
        cout << t2 << " " << fixed << setprecision(6) << g2;

    return 0;
}
```

Problem 8:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, k;
    cin >> n >> k;
    vector<int> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];

    priority_queue<int, vector<int>, greater<int>> pq;
    for (int x : arr) {
        pq.push(x);
```

```
        if (pq.size() > k) pq.pop();
    }

    vector<int> result;
    while (!pq.empty()) {
        result.push_back(pq.top());
        pq.pop();
    }
    sort(result.rbegin(), result.rend());

    for (int i = 0; i < result.size(); i++) {
        if (i) cout << " ";
        cout << result[i];
    }
    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> x(n);
    for (int i = 0; i < n; i++) cin >> x[i];
    int t, s;
    cin >> t >> s;

    for (int i = 0; i < n; i++) {
        int pred = (s * (x[i] - t) <= 0) ? 1 : -1;
        cout << pred << "\n";
    }
    return 0;
}
```

Problem 9:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int v) : val(v), left(nullptr), right(nullptr) {}
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> arr(n);
    for (int i = 0; i < n; i++) cin >> arr[i];
    if (n == 0 || arr[0] == -1) return 0;

    TreeNode* root = new TreeNode(arr[0]);
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;
    while (i < n) {
        TreeNode* node = q.front(); q.pop();
        if (i < n && arr[i] != -1) {
            node->left = new TreeNode(arr[i]);
            q.push(node->left);
        }
        i++;
        if (i < n && arr[i] != -1) {
            node->right = new TreeNode(arr[i]);
            q.push(node->right);
        }
        i++;
    }
```

```
    queue<TreeNode*> bfs;
    bfs.push(root);
    while (!bfs.empty()) {
        int sz = bfs.size();
        TreeNode* last = nullptr;
        for (int j = 0; j < sz; j++) {
            TreeNode* node = bfs.front(); bfs.pop();
            last = node;
            if (node->left) bfs.push(node->left);
            if (node->right) bfs.push(node->right);
        }
        cout << last->val << "\n";
    }

    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

double entropy(const vector<int>& labels) {
    if (labels.empty()) return 0.0;
    int cnt0 = 0, cnt1 = 0;
    for (int y : labels) (y == 0 ? cnt0 : cnt1)++;
    double n = labels.size();
    double p0 = cnt0 / n, p1 = cnt1 / n;
    double h = 0.0;
    if (p0 > 0) h -= p0 * log2(p0);
    if (p1 > 0) h -= p1 * log2(p1);
    return h;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, d;
    cin >> n >> d;
```

```
    vector<vector<int>> features(n, vector<int>(d));
    vector<int> labels(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < d; j++) cin >> features[i][j];
        cin >> labels[i];
    }

    double baseEntropy = entropy(labels);
    int bestFeature = 0;
    double bestIG = -1;

    for (int j = 0; j < d; j++) {
        vector<int> left, right;
        for (int i = 0; i < n; i++) {
            if (features[i][j] == 0) left.push_back(labels[i]);
            else right.push_back(labels[i]);
        }
        double wLeft = (double)left.size() / n;
        double wRight = (double)right.size() / n;
        double condEntropy = wLeft * entropy(left) + wRight * entropy(right);
        double ig = baseEntropy - condEntropy;
        if (ig > bestIG) {
            bestIG = ig;
            bestFeature = j;
        }
    }

    cout << bestFeature << " " << fixed << setprecision(6) << bestIG << "\n";
    return 0;
}
```

Problem 10:

Part A-

```
#include <bits/stdc++.h>
using namespace std;

struct Node {
    int val;
    Node *left, *right;
    Node(int v) : val(v), left(nullptr), right(nullptr) {}
```

```
};

Node* insert(Node* root, int key) {
    if (!root) return new Node(key);
    if (key < root->val) root->left = insert(root->left, key);
    else root->right = insert(root->right, key);
    return root;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    Node* root = nullptr;
    for (int i = 0; i < n; i++) {
        int key; cin >> key;
        root = insert(root, key);
    }

    if (!root) return 0;

    queue<Node*> q;
    q.push(root);
    while (!q.empty()) {
        int sz = q.size();
        for (int i = 0; i < sz; i++) {
            Node* node = q.front(); q.pop();
            cout << node->val;
            if (i < sz - 1) cout << " ";
            if (node->left) q.push(node->left);
            if (node->right) q.push(node->right);
        }
        cout << "\n";
    }

    return 0;
}
```

Part B-

```
#include <bits/stdc++.h>
using namespace std;

struct Split {
    int feature;
    double threshold;
    double gini;
    bool valid;
};

double giniImpurity(const vector<int>& labels) {
    if (labels.empty()) return 0.0;
    int cnt0 = 0, cnt1 = 0;
    for (int y : labels) (y == 0 ? cnt0 : cnt1)++;
    double n = labels.size();
    double p0 = cnt0 / n, p1 = cnt1 / n;
    return 1.0 - p0 * p0 - p1 * p1;
}

Split bestSplit(const vector<vector<double>>& X, const vector<int>& y, const
vector<int>& idx) {
    int n = idx.size(), d = X[0].size();
    Split best{-1, 0.0, 1e9, false};
    for (int f = 0; f < d; f++) {
        vector<pair<double,int>> vals;
        for (int i : idx) vals.push_back({X[i][f], y[i]});
        sort(vals.begin(), vals.end());
        for (int j = 1; j < n; j++) {
            if (vals[j-1].first == vals[j].first) continue;
            double thr = (vals[j-1].first + vals[j].first) / 2.0;
            vector<int> left, right;
            for (auto &p : vals) {
                if (p.first <= thr) left.push_back(p.second);
                else right.push_back(p.second);
            }
            double g = (double)left.size()/n * giniImpurity(left)
                    + (double)right.size()/n * giniImpurity(right);
```

```
            if (g < best.gini) {
                best = {f, thr, g, true};
            }
        }
    }
    return best;
}

int majorityClass(const vector<int>& labels) {
    int cnt0 = 0, cnt1 = 0;
    for (int y : labels) (y == 0 ? cnt0 : cnt1)++;
    return (cnt1 > cnt0 ? 1 : 0);
}

struct Node {
    bool isLeaf;
    int feature;
    double threshold;
    int prediction;
    Node* left;
    Node* right;
    Node(): isLeaf(true), feature(-1), threshold(0), prediction(0), left(nullptr), right(nullptr)
{}
};

Node* buildTree(const vector<vector<double>>& X, const vector<int>& y, const
vector<int>& idx, int depth) {
    Node* node = new Node();
    if (depth == 2 || idx.empty()) {
        node->isLeaf = true;
        vector<int> labels;
        for (int i : idx) labels.push_back(y[i]);
        node->prediction = majorityClass(labels);
        return node;
    }
    Split sp = bestSplit(X, y, idx);
    if (!sp.valid) {
        node->isLeaf = true;
        vector<int> labels;
        for (int i : idx) labels.push_back(y[i]);
```

```
        node->prediction = majorityClass(labels);
        return node;
    }
    node->isLeaf = false;
    node->feature = sp.feature;
    node->threshold = sp.threshold;
    vector<int> leftIdx, rightIdx;
    for (int i : idx) {
        if (X[i][sp.feature] <= sp.threshold) leftIdx.push_back(i);
        else rightIdx.push_back(i);
    }
    node->left = buildTree(X, y, leftIdx, depth+1);
    node->right = buildTree(X, y, rightIdx, depth+1);
    return node;
}

int predict(Node* root, const vector<double>& x) {
    Node* node = root;
    while (!node->isLeaf) {
        if (x[node->feature] <= node->threshold) node = node->left;
        else node = node->right;
    }
    return node->prediction;
}

void printTree(Node* node, int depth=0) {
    if (node->isLeaf) {
        cout << string(depth*2, ' ') << "Leaf: predict=" << node->prediction << "\n";
    } else {
        cout << string(depth*2, ' ') << "Split: feature=" << node->feature
             << " threshold=" << node->threshold << "\n";
        printTree(node->left, depth+1);
        printTree(node->right, depth+1);
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
```

```
    int n, d;
    cin >> n >> d;
    vector<vector<double>> X(n, vector<double>(d));
    vector<int> y(n);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < d; j++) cin >> X[i][j];
        cin >> y[i];
    }

    vector<int> idx(n);
    iota(idx.begin(), idx.end(), 0);
    Node* root = buildTree(X, y, idx, 0);

    printTree(root);

    int correct = 0;
    for (int i = 0; i < n; i++) {
        if (predict(root, X[i]) == y[i]) correct++;
    }
    double acc = (double)correct / n;
    cout << "Accuracy: " << fixed << setprecision(6) << acc << "\n";
    return 0;
}
```